# History of JAVA: -

* In **1990 Sun micro system** try to build set up box. They assign to Green Team. **James Gosling** is the team leader of that team is also known as the **father of the Java.**
* Earlier name of Java is known as OAK(OAK Tree represent to strength)

Facts of OAK-

* It is the National Tree of USA, Italy and Numbers of country etc.
* A company named OAK technology is already present in USA. So the founder of OAK language change its name to JAVA( which represent the location java island in Indonesia where, coffee 1st discovered)

# Version hierarchy of JAVA:

1. JDK 1.0------->1995
2. JDK 1.2------->1997

All are the Stable Version of Java

1. JDK 1.5------->2004
2. JDK 1.8------->2014

Here we are going to use JDK 1.8

1. JDK ……………
2. JDK 15-------->2020
3. JDK 17-------->2021

# What is JAVA: -

* Java is a general-purpose programming language.
* General purpose means it is used in both hardware and software Industries.
* But JAVA is very much famous in software industry to develop the application.

# Features of JAVA: -

1. It is simple.
2. Secure
3. Platform Independent
4. Portable
5. Robust
6. Object oriented programming language
7. Class structure programming language
8. Type safe programming language
9. Interpreted programming language
10. WORA language (Write once run anywhere)

# Type of Application: -

1. The application which can run without internet called 🡪 Web Application.
2. The application which can run without internet called 🡪 Standalone Application.

# Platform of JAVA: -

There are 3 types of platforms in Java. -

# JDK Architecture-

execution (JRE)

Coding

Pre define syntax library

JVM

Compilation

x.class

Byte code (0,1)

X.Java

X.Java

**JRE**

**JDK** code (0,1)

**JDK**-

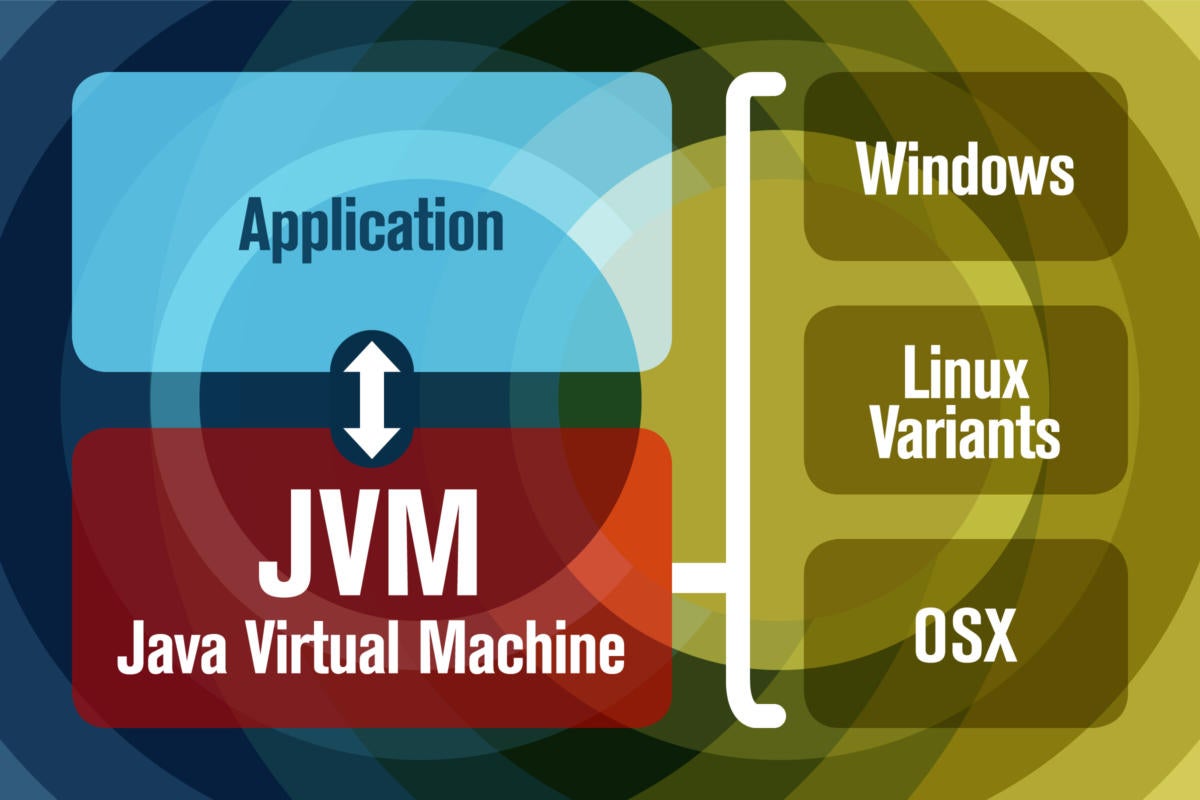
If **jdk compilation is fail** then jdk will show **Fully qualified Information will show**.

1. **Where is the error?**
2. **What is the error?**
3. **How is the error?**
4. **How many error?**

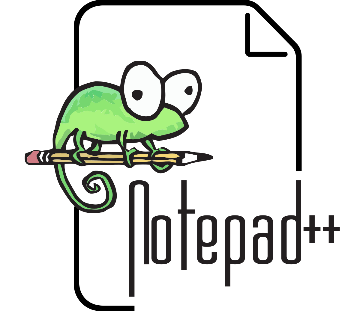
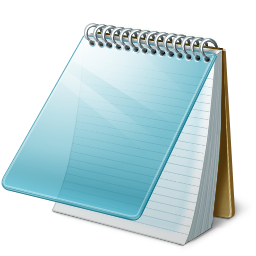
* It stands for java development kit.
* It provides the environment for coding & compilation.

**JRE**-

* It stands java run time environment.
* It provides the environment for execution.

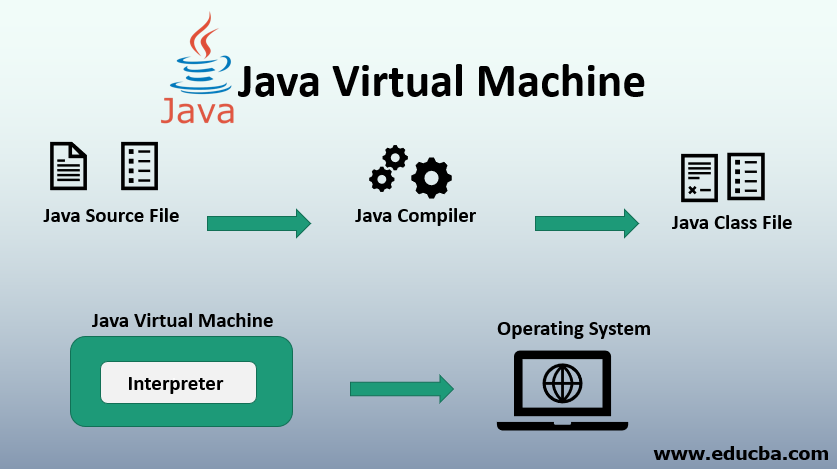
**JVM**-

* It stands for java virtual machine.
* JRE is just a folder name inside that a powerful machine is present that is JVM.
* That JVM is fully responsible for the execution of Java Program.

CODING –

* It is done by programmer.
* Usually programmer will write the code in editors.

Ex- notepad, notepad++, edit+

* Usually programmer done with the program save the file with ‘file\_name.java’ extension.

Compilation –

* It is done by compiler.
* It is the process of cross-checking ‘**file\_name.java**’ file with predefine syntax library to encounter the syntax level mistake.

## Error Case-

1. Compilation pass
2. Compilation fail

**Compilation pass-**

* If compilation pass than ‘**file\_name.java**’ file convert to ‘**class\_name.class**’ file contain ‘0’ and ‘1’. This is known as machine level language.
* Computer can understand only machine level language. Because of this ‘**class\_name.class**’ file become portable, platform independent, secure & WORA language.

**Compilation Fail-**

* If the compilation become fail then ‘**file\_name.java**’ file convert to ‘**class\_name.class**’ and we will get a compile time error.
* This compile time error contain Fully Qualified Information about the error(**FQI**).

**FQI-** FQI means we will know the following question answer.

* **What is the error?**
* **Where is the error?**
* **How many error?**
* **How to resolve the error?**

**Example**-

Class Test

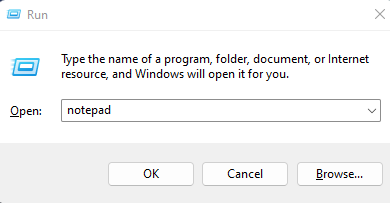
{

Public static void main(string[] args)

{

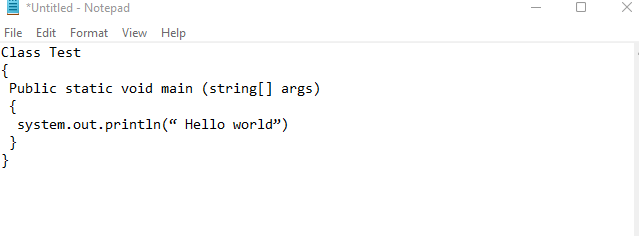
system.out.println(“ Hello world”);

}

}

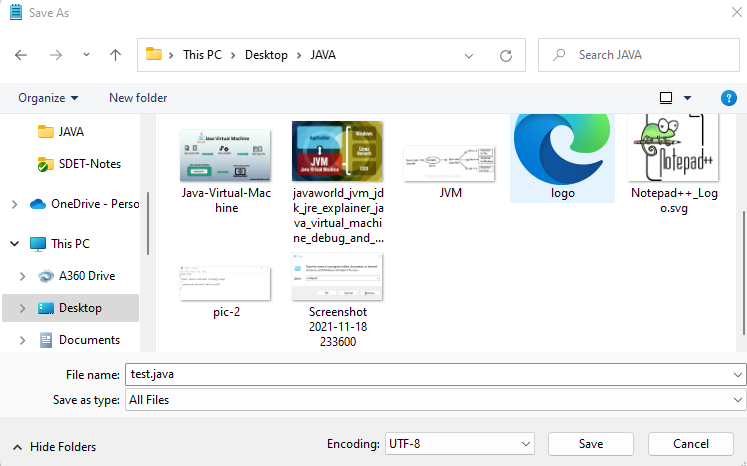
**For execution Process**-

* To open notepad

1. Win+R
2. Enter
3. Type ‘notepad’ in run

* To save the file-

1. File
2. Got to Save option
3. Save file\_name as ‘filename.java’
4. Change to all files, then click the save option.



**File\_name.java**

**File type ‘All type**

The **compilation and execution** takes place in system console or **command promt.**

**To open command promt and execution process –**

1. Win+R->
2. Type ‘cmd’
3. Ok/Enter

**Condtion: -**

1. **With in same Driver**

(cd- change directory, here from D to D/ODM18

Cd folder path

Example =

1. **With in same Driver**

(cd- change directory, here from D to D/ODM18

Cd folder path

Example =

* **Code for execution: -**

**Syntax- javac file\_name.java**

* **Code for execution: -**

**Syntax- java class\_name**

**Case-1->**

**Program:-**

class Test

{

public static void main(String [ ] args)

{

system.out.println(“ Hello world”);

}

}

**Code for compilation-**

* **Ex- javac sample.java**

**Folder Name**

**Code for execution-**

* **Ex- java Demo**

**Variable & methods**

**Note:-**

* **To avoid the future execution according to java coding standard always proving same file name & variable & methods.**

**Case-2->**

**Program:-**

**Compilation – Pass ✔**

**Execution – fail ❌**

Class A

{

}

**Error- mainmethod not found in class A**

**Note:-**

* **When we get error at execution failed it is runtime error.**
* **Without the mainmethod we can compile a java program.**
* **If you want to execute a java program then mainmethod is compulsory.**

**Case-3->**

**Program:-**

Class A

{

}

Class B

{

}

Class C

{

}

**Note:-**

* **In a class we can write n.. numbers of class and after compilation, it will**

**Create separate class for each class.**

**Case-4->**

**Program: -**

Class A

{

}

Class A

{

}

Class A

{

}

**Note: -**

* **In a class we can write one variable & methods (Variable & methods are case-sensitive and always start with upper case).**

# Class Structure of Java-

class Demo

keyword- Always start with **lowercase**

Identifier- May be **UPPERCASE** May be **lowercase**

Keyword Identifier

{

Variables

Methods

Class Member

Blocks

Constructs

}

public static void main(String [] args)

{

system.out.println(“Hai”);

}

O/p- Hai

**Java Class-**

* Class is an area where we declare class member i.e. variables, methods, block and constructors.
* Entire java program we will write inside the class only.
* If we write outside the java class then we will get compile time error.
* To execute a java mainmethod is mandatory.

**Keywords-**

* Keywords are the important words which have the functionality in a short manner.
* In **java** there are **total 53 keywords** present.
* **All keywords** we have to write in **lower case**.

**Identifiers-**

* These are used to identify the class or class members.
* Identifiers may exist in upper case and lower case also.
* Identifiers may be predefined / user defined.

## Rules to provide the variable & methods

1. The variable & methods always start with upper case.

Example – class Demo**✔** Example – class demo **❌**

1. Numeric Alpha variable & methods not allowed.

Example – class 21demo **❌**

1. Alpha Numeric variable & methods allowed.

Example – class demo12 **✔**

1. Space is not allowed in between variable & methods.

Example – class de mo **❌**

1. Special character not allowed except ‘$, \_’ in variable & methods.

Example – class Demo\_02**✔** Example – class demo@02 **❌**

## Rules to provide the variable & methods

1. The variable & methods always start with lower case.

Example – main() **✔** Example – Main() **❌**

1. Numeric Alpha variable & methods not allowed.

Example – 1main() **❌**

1. Alpha Numeric variable & methods allowed.

Example – sum1 () **✔**

1. Space is not allowed in between variable & methods.

Example – sum one () **❌**

1. Special character not allowed except ‘$, \_’ in variable & methods.

Example – sum\_one () **✔** Example – sum@one () **❌**

**Note -**

* **Always we should start our class name then it should always start with upper case & method should in lower case.**

**Example- class Demo✔ class demo ❌**

**Method ( ) ❌ method ( )✔**

## Rule Printing Statement in java

* Printing statement are used to print the output in the console.
* There are two printing statements are in java.

**Ex-** 1.system.out.println ()

2. system.out.print ()

**System.out.print ()**

**Case-1**

* It is used to print the output in the console and keep the courser in the same line.

class Demo

{

public static void main(String [ ] args)

{

system.out.print(“hello”);

system.out.print(“world”);

}

}

O/p- helloworld

**Case-2**

* In this case of system.out.print printing statement is compulsory. Otherwise we will get compile time error.

class Demo

{

public static void main(String [ ] args)

{

system.out.print(“hello”);

system.out.print( );

}

}

O/p- compile time error

**System.out.println ()**

**Case-1**

* It is used to print the output in the console and keep the courser in the next line.

class Demo

{

public static void main(String [ ] args)

{

system.out.println(“hello”);

system.out.println(“world”);

}

}

O/p- hello

world

**Case-2**

* In this case of system.out.println printing statement is not compulsory.

class Demo

{

public static void main(String [ ] args)

{

system.out.println(“hello”);

system.out.println( );

}

}

O/p- hello

## Rules for Print Statement for number

**Case-1**

class Test

{

public static void main(String [ ] args)

{

system.out.println(“100”);

O/p- 100

(it will consider ‘1’,’0’,’0’ each as individual character

}

}

**Case-2**

class Test

{

public static void main(String [ ] args)

{

O/p- 100

(it will consider 100 as Number

system.out.println(100);

}

}

**Case-3**

class Test

{

public static void main(String [ ] args)

{

O/p- Compilation time error

system.out.println(hello);

}

}

**Case-4**

class Test

{

public static void main(String [ ] args)

{

system.out.println(“A”);

O/p- A (here “A” as string)

}

}

**Case-5**

O/p- A (here ‘A’ as charater)

class Test

{

public static void main(String [ ] args)

{

system.out.println(‘A’);

}

}

# Operator Overloading

**Case-1**

class **Test**

{

public static void main(String [ ] args)

{

O/p- Hello --> String

123 --> String

123 --> Number

System.out.println(“Hello”);

System.out.println(“123”);

System.out.println(123);

}

}

**Case-2**

class **Test**

{

**public static void** **main**(**String** [ ] args)

{

**System.**out**.println**(12+12);

O/p- 24 --> String

12+12 --> String

1212 --> Number

**System.**out**.println** (“12+12”);

**System.**out**.println** (“12”+”12”);

}

}

***\*Note-***

1. ‘+’ Sign with Numbers 🡪 Addition
2. ‘+’ Sign within String🡪 Nothing (Print as it is )
3. ‘+’ Sign with String 🡪 Concatenation🡪Joining of 2 information is called concatenation.

O/p- 121212 --> String

**Case-3**

class **Test**

{

**public static void** **main**(**String** [ ] args)

{

System.out.println(“12”+12+12);

}

**JAVA reads statement left to right (String + Number= String)**

O/p- 121212 --> String

}

1. If any one information is string it will perform throughout concatenation.

**Case-4**

class **Test**

{

**public static void** **main**(**String** [ ] args)

{

**System.**out**.println**(12+12+”12”);

}

**JAVA reads statement left to right**

O/p- 2412 --> String

}

***\*Note-***

* Whenever ‘+’ sign perform concatenation & addition at a same time then it is called as operator overloading.
* ‘+’ changes its behavior is nothing but operator overloading.

**Case-5**

class **Test**

{

**public static void** **main**(**String** [ ] args)

{ **System.**out**.println**(“Hello”+”World”);

**System.**out**.println** (“Hi”+15+30);

O/p- HelloWorld 🡪

Hi1530 🡪

45Hi 🡪

Hi45🡪

**System.**out**.println** (15+30+Hi);

**System.**out**.println** (“Hi”+(15+30));

}

}

**Case-5**

Concatenation help to give meaningful complete information

class **Test**

{

**public static void** **main**(**String** [ ] args)

O/p- 45

Addition of 15 and 20 is: 45

{

**System.**out**.println** (15+30);

**System.**out**.println** (“Addition of 15 and 20 is: “+(15+30));

}

If in the below program if we use variable instead fix value then we have the flexibility to use the program for multiple times.

}

# Variable

class **Demo**

{

**public static void** **main**(**String** [ ] args)

{

**System.**out**.println** ((“Addition is: “+(15+5));

**System.**out**.println** (“Subtraction is: “+(15-5));

**System.**out**.println** (“Multiplication is: “+(15\*5));

}

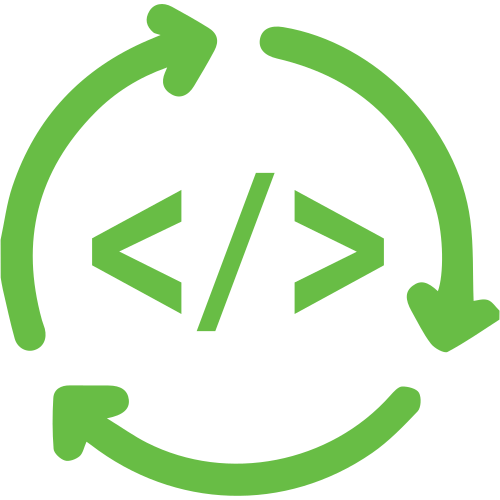
O/p-  **Addition is: 20**

**Subtraction is: 10**

**Multiplication is: 75**

}

## Why we use variable?

* In a worldwide programming field dealing directly with the data/program is not at all recommend approach.
* This is because any kind of data is not a onetime use content.
* To overcame this situation in every programming language variable exist.
* Variable are nothing but a piece of memory which is use store a data.



Info

Why we should go for variable?

1. To avoid hard coding of data.
2. Maintenance of code easier.
3. We can achieve information reusability.

* Before declaring variable programmer need to be design which types of data variable can store.
* It is compulsory to decided which type of data.
* Base on data type our code behave differently.

## Data Types-

* Data types is used to provide instruction to variables regarding what type of data it can store.

There are 2 types of data type

1. Primitive Data type.
2. Non-Primitive Data type.

## Primitive Data Type-

* The data type which is predefine in java in the form of keyword is called primitive data type.
* All primitive data type are fixed in their size.
* As all primitive data type are keyword, so they are compulsory to declared in lower case.

## Range of Data Type-

* Formula = No of value we can store is = 2Bit.
* Example = Byte memory is 1byte. So it can store up to 28 no of values that is nothing but 256 nos. Then Range will be (2Bit/2) =(28/2)=(256/2)= (-128 to 127)

## Non-Primitive Data Type-

* Based on popularity level there are 2 types of non-primitive data type.

1. Class/Interface🡪 In term of technical aspect.
2. String 🡪 In term of popularity in the sector.

* The Data type is not predefined in java as a keyword is consider as non-primitive data type.
* Every non primitive data type is identifier.
* All non-primitive data type are not fixed in their size.

## How to declare a variable

Data Type ref name= Value/Information

Int num;} variable Declaration.

Allocation name.

Memory cell will generate

num=20;} variable Initialization.

num

20

**Declaration**

* Same @ a time 🡪 int x = 20;

**Declaration**

## Re-initialization

Int x; 🡪 Declaration X🡪

**20**

X=20 🡪 Initialization X🡪

**25**

X=25 🡪 Re-initialization X🡪

## Type of variable

1. Local variable
2. Static variable
3. Non-static variable

## Local variable

Example-

class **Test**

{

O/p-

**public static void** **main**(**String** [ ] args)

{ int x=10;

**System.**out**.println** (X);

}

Declaring a variable inside a methods, Block, constructor is called as **local variable**

}

class **Test**

{Method

{ int x=10;

}

Block

{ int y=20;

}

Constructor

{ int z=30;

}}

* Local variable is accessible only in that particular area where it is declared.

Example🡪

class **Test**

O/p- error variable X might not been initialized.

{

**public static void** **main**(**String** [ ] args)

{ int x;

**System.**out**.println** (X);

}

}

Example🡪

class **Test**

{

Hello

**public static void** **main**(**String** [ ] args)

{ int x;

**System.**out**.println** (“Hello”);

}

}

* Each local variable compulsory to initialization before utilization.
* Local variables don’t have any default value.

Example🡪

class **Test**

{

**public static void** **main**(**String** [ ] args)

{ int x=10;

int y=20;

int z=x+y;

O/p- z valu System.**out**.println **(“y value is: ”+y);**

e is :30

**System.**out**.println** (“z value is: ”+z);

}

}

## Static variable-

* Declaring a variable outside a Member or Block or Constructer with ‘static’ keyword is called as static variable.

class **Test**

{

static int x=10;

**public static void** **main**(**String** [ ] args)

{

}

}

Example-

class **Test**

{

static int x=10;

static int y=5;

**public static void** **main**(**String** [ ] args)

{

**System.**out**.println** (“x value is: ”+x);

System.**out**.println **(“y value is: ”+y);**

System.**out**.println **(“product value is: ”+(x\*y));**

**}**

**}**

O/p- x value is 10

Y value is 5

Product values is 50

* Static variable is accessible in entire class as well as in other class as we.
* Due to this static variable is consider as permanent memory variable.

Example🡪

class **Test**

{

static int x=10;

method

{

}

Block

{

}

Constructor

{

}

**}**

* **We can’t use the static inside any Method, Block, Constructor**

Example🡪

class **Test**

{

**public static void** **main**(**String** [ ] args)

{

static int x=10;

O/p- error 🡪 illegal start of expression

}

}

* **Local variable can’t start static keyword.**

Example-

class **Test**

{ static int x=10;

**public static void** **main**(**String** [ ] args)

{

**System.**out**.println** (x);

}

}

class **Demo**

{

**public static void** **main**(**String** [ ] args)

{

**System.**out**.println** (x);

}

O/p- error: cannot find symbol

System.out.println(x);

}

## Declaring a variable in different class

class **Test**

{ static int x=10;

**public static void** **main**(**String** [ ] args)

{

**System.**out**.println** (x);

}

}

class **Demo**

{

**public static void** **main**(**String** [ ] args)

{

O/p- 10

10

We can access static variable anywhere in a project by using class name.

**System.**out**.println** (Test.x);

}

**}**

**classname.variable**

**Syntax🡪**

## Features of static variable

* Static variable is not compulsory to initialized.
* Static variable has default values based on data type.
* Every data type will differently default value.

## Static variable/Non-static variable default vaule

Byte 🡪0

Shot🡪0

int🡪0

long🡪0 (long value always ends with ‘l’)

float🡪0.0(float value always ends with ’f’)

double🡪0.0

char🡪Empty space

boolean🡪false

string🡪Null

class🡪Null (example class name variable name🡪test.x)

## How to write /declar long or float data?

Example –

class **Demo**

{ static float a=24.54f;

**public static void** **main**(**String** [ ] args)

{

**System.**out**.println** (a);

}

**}**

*Priority goes to local variable-*

1. static & local variable can have same name.
2. In such case 1st priority always goes to local.

Example-

class **Demo**

{ static int x=10;

**public static void** **main**(**String** [ ] args)

{ int x=100;

O/p- 100

**System.**out**.println** (x);🡪

}

**}**

class **Demo**

O/p- 10 100

{ static int x=10;

**public static void** **main**(**String** [ ] args)

{

If static variable & local variable have same name then static variable need to be access by class name explicitly

int x=100;

**System.**out**.println** (x);

**System.**out**.println** (Test.x);

}

10

20

30

40

**}**

static

Local

**Case-1**

class **Test**

O/p-

**10**

**20**

**30**

**40**

**60**

{ static int x=10;

**public static void** **main**(**String** [ ] args)

{

* GO to static variable =10
* Re-initialized as 20
* We will initialize

x as local=30.

* Re-initialized as 40
* Final value 40+20=60

**System.**out**.println** (x);

x=20;

**System.**out**.println** (x);

Int x=30;

**System.**out**.println** (x);

x=40;

**System.**out**.println** (x);

**System.**out**.println** (x+Test.x);

} }

## Effect of static variable-

class **Test**

O/p- 10 50

{ static int x=10;

**public static void** **main**(**String** [ ] args)

{

**System.**out**.println** (x);

x=50;

**System.**out**.println** (x);

}

**}**

class **Demo**

O/p- 10

{

**public static void** **main**(**String** [ ] args)

In this case for class Demo static variable test.x will call the initial value of the static variable at class test

{

**System.**out**.println** (Test.x);

}

**}**

* Re-initialization of static variable will affect only in that particular class.
* If we try to fetch static variable from another class then we will get that value which we provide at the time of declaration.
* As static variable can access on anywhere by using class name so it is considered as class variable.

## Non-Static variable-

* Declaring a variable out side member or Block or constructer without static keyword is called as non-static variable.

Main method is always static & never change

class **Test**

NSV-***Non Static variable (without static keyword outside member***

Out

{

int x=10;-🡪

**public static void** **main**(**String** [ ] args)

{

**Static area allow only static variable**

**Non- static variable cant be reffered from static context/Area**

**System.**out**.println** ();

}

**}**

***Example-***

class **Test**

{

int x=10;-🡪

**public static void** **main**(**String** [ ] args)

{

**System.**out**.println** ();

}

**}**

NSV-***Non Static variable (without static keyword outside member***

Out

***Example-***

class **Test**

{ int x=10;

Here we can compile but we can’t execute we will run time error

**public void** **main**(**String** [ ] args)

{

**System.**out**.println** ();

}

**}**

## How to excess non static member in static Area?

* By creating a Object.

## Object –

* Object is nothing but a new repetitive of class through object we can excess any class member.

**\*Note-**

1. Object is never for a member it is always for a class.

2. Static member excess anywhere by object creation.

3. Apart from that non-static & static variable have same properties.

For example 🡪Non-static member is permanent variable.

Non-static /static is not compulsory to initialized.

Because they have default value based on data type.

Non-static variabele can use in the entire class as well as other class.

## Object creation-

* **Object of a class** called as **instance of a class** .
* Due to this **non-static variable** is also called as **instants variable**.

1. In java ,Object is always create by **new** keyword

new;

It will show compile time error( Idenitifier expected)

1. Always we have to initialize the object

*Syntax*🡪**new class\_name ();**

`Example🡪 new Test();

**X🡪10**

**Object for test**

Object is created for Test class & all non-static member load inside new objects

1. Providing Reference to Object

**Test t = new Test();**

Syntax-class name reference name= new class\_name ();

**Reference Name**

**New empty memory generate in JVM**

**Reference Name**

**Non-primitive data type**

Example-

class **Test**

{

int x=10;

**public static void** **main**(**String** [ ] args)

{

Test t= new Test();

x🡪10

**System.**out**.println** (t.x);

}

**}**

**Test Object**

**O/p- 10**

class **Test**

{

int x=10;

int y=20;

**public static void** **main**(**String** [ ] args)

x🡪10

y🡪20

{

Test t= new Test();

**System.**out**.println** (t.x+t.y);

}

**}**

O/p-30

**Test Object**

**Static variable always has default value.**

*Example🡪*

class **Test**

{

int x=10;

int y;

**public static void** **main**(**String** [ ] args)

x🡪10

y🡪20

{

Test t= new Test();

**System.**out**.println** (t.x+t.y);

}

**}**

O/p-0

O/p-30

## Calling non static variable to another class-

*Example🡪*

class **Test**

{

int x=10;

**public static void** **main**(**String** [ ] args)

{

Test t= new Test();

**System.**out**.println** (t.x);

O/p-10

}

**}**

class **Demo**

{

**public static void** **main**(**String** [ ] args)

{

Test t= new Test();

**System.**out**.println** (t.x);

O/p-10

O/p-10

}

**}**

*Example🡪*

class **Test**

{

int x=10;

**public static void** **main**(**String** [ ] args)

{

Test t= new Test();

T1

~~10~~

100

O/p

10

10

100

10

100

200

**System.**out**.println** (t1.x);

**System.**out**.println** (t2.x);

t1.x=100;

**System.**out**.println** (t1.x);

**System.**out**.println** (t2.x);

T2

~~10~~

200

t2.x=200;

**System.**out**.println** (t1.x);

**System.**out**.println** (t2.x);

}

**}**

***\*Note-***

* Non-static variable or member will have multiple copy in a memory based on object creation.
* Reinitialization of non-static variable of a only in that particular object.

## Difference between Local vs Static vs Non-Staic-

|  |  |  |
| --- | --- | --- |
| **Local** | **Static** | **Non-static** |
| Declare Inside Member or Block or Constructor. | Declare Outside Member or Block or Constructor with static keyword. | Declare Outside Member or Block or Constructor with static keyword. |
| Access only within a area where it is declared. | Access anywhere in project by Class Name. | Access anywhere in project by Object Creation. |
| It is temporary memory variable. | It is Permanent Change. | It is Permanent Change. |
| Compulsory to initialized. | Not compulsory to initialized. | Not compulsory to initialized. |
| It will not have default value. | Default value based on datatype. | Default value based on datatype. |
| It will have only one temporary copy. | It will have only one permanent copy. | It will have multiple copy based on object creation. |
| - | It is also called class variable. | It is also called Instance variable. |
| Access directly. | Access directly or class Name or Object creation. | Access only by Object creation. |
| It will load in stack Area. | It will load in method Area. | It will load in Heap Area. |

# Operators-

* Types of Operator-

|  |
| --- |
|  |

## Arihtmetic Operator-

|  |  |  |
| --- | --- | --- |
|  | | |
| ***Operator*** | ***Use*** | ***Description*** |
| + | X + Y | Adds op1 and Y; also used to concatenate strings |
| - | X - Y | Subtracts Y from op1 |
| \* | X \* Y | Multiplies X by Y |
| / | X / Y | Divides X by Y |
| % | X % Y | Computes the remainder of dividing X by Y |

class **Test**

{

int x=10;

int y=3;

**public static void** **main**(**String** [ ] args)

{

O/p

13

7

30

5

1

**System.**out**.println** (x+y);

**System.**out**.println** (x-y);

**System.**out**.println** (x\*y);

**System.**out**.println** (x/y);

**System.**out**.println** (x%y);

}

**}**

## Unary Operator-

* The operator which use to perform **incrementation or decrementing by 1** is called as **unary operator**.
* These operators perform action on unit place.
* These are 2 unary operators present in Java with four cases.

**Int x=10;**

**System.out.println(x);**

**--x;**

**System.out.println(x);**

9

10

Int x=10;

System.out.println(x);

++x;

System.out.println(x);

10

11

**Int x=10;**

**System.out.println(x);**

**X++;**

**System.out.println(x);**

9

10

**Int x=10;**

**System.out.println(x);**

**X--;**

**System.out.println(x);**

10

11

## Example with various cases-

***P****re-increment* *Pre-decrement*

10

10-1=9(1st it will decrease with 1 and print/store values of x in memory.

9

Int x=10;

System.out.println(x);

System.out.println(--x);

System.out.println(x);

Int x=10;

System.out.println(x);

System.out.println(++x);

System.out.println(x);

10

10-1=9(1st it will utilize values of x then decrease with 1.

9

10

1+10=11(1st it will increase with 1 and print/store values of x in memory.

11

*Post-increment*

**Int x=10;**

**System.out.println(x);**

**X++;**

**System.out.println(x);**

10

1+10=11(1st it will utilize values of x then increase with 1.

11

*Post-decrement*

**Int x=10;**

**System.out.println(x);**

**X--;**

**System.out.println(x);**

*\*Note-*

* In case of post operation 1st it will utilize old value & than operation perform.
* In case of pre-operation 1st it will perform operation & then it will utilized.

***Case-1***

class **Test**

{

int x=10;

int x=13;

**public static void** **main**(**String** [ ] args)

O/p-

10

13

10

{

**System.**out**.println** (x++ + ++x - --y – y-- + x++);

**System.**out**.println** (x);

**System.**out**.println** (y);

}

**}**

***Case-2***

class **Test**

{

**public static void** **main**(**String** [ ] args)

O/p-

10

13

10

{

int x=10;

int x=x++;

**System.**out**.println** (x);

O/p-

11

10

**System.**out**.println** (y);

}

**}**

***Case-3***

O/p-

Error: unexpected type ++(++x),

Required variable

Found: value

class **Test**

{

**public static void** **main**(**String** [ ] args)

{

int x=10;

++(x++);

Unary operator always used with variables ++11 🞮

**System.**out**.println** (x);

O/p-

11

10

}

With constant we can’t use unary operator .

**}**

***Case-4***

O/p-

Error: unexpected type ++(++x),

Required variable

Found: value

class **Test**

For alphabet it will (++) increase to next alphabet.

So we can use unary operator for charater.

{ **public static void** **main**(**String** [ ] args)

{

Char ch=“B” ;

Char ch1=ch++;

**System.**out**.println** (ch);

O/p-

11

10

}

With constant we can’t use unary operator .

}

* Unary operator can be used with character.

***Unary operator for float-***

***Case-5***

class **Test**

{ **public static void** **main**(**String** [ ] args)

{

double d=“3.14” ;

O/p-

4.140000001

4.140000001

double d=++d;

**System.**out**.println** (d);

**System.**out**.println** (d1);

}

}

***Case-6***

class **Test**

{ **public static void** **main**(**String** [ ] args)

{

double b=true;

O/p-

Error: Bad operand type Boolean for unary operator’++’

O/p-

4.140000001

4.140000001

b++;

**System.**out**.println** (b);

**System.**out**.println** (d1);

}

}

***Case-7***

**As byte has range from -127 to 128, if we increase value by x++ then it will go in a cyclic manner**

class **Test**

{ **public static void** **main**(**String** [ ] args)

{

byte b=127 ;

O/p-

-128

Byte Range

b++;

**System.**out**.println** (b);

}

}

***Case-8***

class **Test**

**Int to byte conversion not possible**

{ **public static void** **main**(**String** [ ] args)

{

byte b=127 ;

O/p-

-128

Byte Range

b=b+1;

**System.**out**.println** (b);

}

O/p-error incompatible types.

}

# Assignment Operator

* There are 2 type of operator in assignment operator.

1. Normal Operator 🡪 **‘ = ’** example…**int x=10🡪 here 10 is assign to x.**
2. Compound operator🡪 **‘ += ’** example…**int x+=10 => x=x+10=>x=20**

**‘ -= ’** example…**int x-=10 => x=x-10=>x=0**

**‘ \*= ’** example…**int x\*=10 => x=x\*10=>x=100**

**‘ /= ’** example…**int x/=10 => x=x/10=>x=1**

**‘ %= ’** example…**int x%=10 => x=x%10=>x=0**

***Case-1***

class **Test**

{ **public static void** **main**(**String** [ ] args)

{

int x=10 ;

int x=10 ;

x\*=y; // x = x\*y=10\*5=50

O/p-

50

5

**System.**out**.println** (x);

y%=x; // y = y%x=5\*50=5

**System.**out**.println** (d1);

}

}

# Relational Operator

* These operators are always performed **compression between 2 data**.

|  |  |
| --- | --- |
| **Operator Symbol** | **Operator Symbol** |
| **<** | Less than |
| **>** | Greater than |
| **<=** | Less than equal |
| **>=** | Greater than equal |
| **==** | Double equal |
| **!=** | Not equal to |
| **!** | Not |

**System.**out**.println** (100>50);

True

**System.**out**.println** (50<60);

True

**System.**out**.println** (50!=50);

**False**

* **All relational operator always returns Boolean value is True/False.**

***Case-1***

class **Test**

{ **public static void** **main**(**String** [ ] args)

{

int x=10;

O/p-

10 >8 (True)

10 !=9 (True)

10 = 9 (False)

11

10

int x=9;

**System.**out**.println** (x++ > --y);

**System.**out**.println** (++x != ++y);

**System.**out**.println** (x++ == y++);

**System.**out**.println** (x);

**System.**out**.println** (y);

}

}

***Case-2***

class **Test**

{ **public static void** **main**(**String** [ ] args)

O/p-

False

True

{

**System.**out**.println** (‘B’>’M’); 🡪 False

**System.**out**.println** (‘B’<’M’); 🡪 True

}

}

***Case-3 (Type Varient)***

class **Test**

{ **public static void** **main**(**String** [ ] args)

O/p-

False

True

{

**System.**out**.println**(90>60>30);

O/p-

Error 🡪Bad operation type.

} True>30 int

} Boolean

***Case-4 (Comparing Character)***

class **Test**

{ **public static void** **main**(**String** [ ] args)

O/p-

False

True

{

Error 🡪Bad operand types

for binary operator

**System.**out**.println**(“suryavanshi”> ”singham”);

}

}

## Not!

!(NOT)= It will convert true value into false or false value into true.

!(True)🡪 False

!(False)🡪 True

***Example***

***Case-4 (Comparing Character)***

class **Test**

O/p-

False

{ **public static void** **main**(**String** [ ] args)

{

**System.**out**.println**(!(“50”== ”50”));

}

}

## Complimentry operator-

* In java ‘~’ denotes complimentary operator.

***Case-4 (Comparing Character)***

class **Test**

O/p-

-17

88

O/p-

False

{ **public static void** **main**(**String** [ ] args)

{

int x =16;

**System.**out**.println**(~x);

int y =-89;

**System.**out**.println**(~x);

} }

## Ternary Operator-

Syntax

**Value= Condition? Value for true: value for false**

***Case-1***

int z=10>5? 100:20;

sopln(z); 🡪100

***Case-2***

int x=10 != 10? 5:20;

sopln(x); 🡪20

* As this **operator changes their value** **based on condition** it is also called as conditional operator.